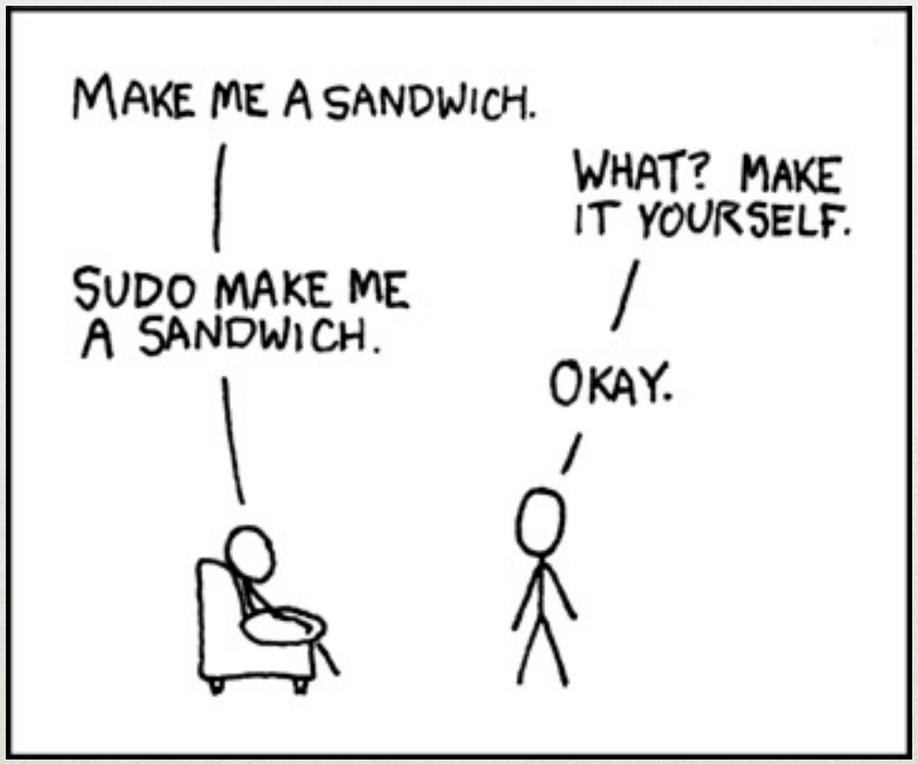
Internet Programming in Python - Week 4

Live on the internet



http://xkcd.com/149/

Brian Dorsey
brian@dorseys.org

A moment to reflect

I am a moron.

This section from: http://www.cimgf.com/2009/03/26/dont-blindly-trust-debb/ -- Marcus Zarra

Documentation

Examples

Blogs

Books

D E B B

always outdated

even when coding alone
there are three people
involved

ME

First, there is me.
I am a clever and talented programmer. I know to keep my code clean and easy to read. I try to do things the right way and I know that if old code is wrong that I am better off fixing it now then trying to code around it.

PAST ME

Second, there is past me. This guy is a fracking moron. He pisses me off on a daily basis as I am having to constant re-write his code. He is generally clueless and causes me more headaches than anyone else.

FUTURE ME

Finally there is the future me. This guy is smarter than I am, has seen more than I have and has done more. My goal, in addition to making sure my code works, is to make sure that my intentions are clear for him so that I don’t piss him off too badly.

this is why your
old code looks terrible
(hopefully)

Questions and Review (20)

some thoughts from the assignments

Assignment

Choose one of the following:

- Write a program which accesses information from two web APIs and combines the data.
- Find and follow each of the links on the course syllabus, download the content and save it into a folder by week. Choose useful filenames.

  Also: set aside time to experiment with the commands from the reading for next week.

A

UW Certificate Program in **Python Programming**

Internet Programming in Python

Winter Term, 2011 - (10 Sessions) Tuesdays 6 - 9 pm, January 11 through March 15

This program also includes **Programmming in Python** (Autumn 2010) and **System Development with Python** (Spring 2011).

Summary. Readings. Schedule.

UPDATES: Course announcements and updates will be posted **here**.

Course objectives: This course emphasizes distributed programs and web applications - how they work and how to program them in Python. Students will explore the underlying principles and their expression in Python libraries. Students will learn contrasting approaches in creating applications: programming with the low-level libraries versus using highly integrated frameworks. All topics will be presented with a focus on solving real problems with simple, pragmatic code.

Prerequisites: Students should have successfully completed **Programmming in Python** or have an equivalent level of experience. Contact the instructor prior to registering if not in the certificate program.

Textbooks: Various online sources will be used.

Instructor: Brian Dorsey.
Brian Dorsey is a database and Python developer with over 10 years of experience using Python professionally. He currently works at Vulcan Inc., developing command line tools, simple web applications, Windows services, HTTP/JSON APIs and the occasional iPhone prototype. He is excited about information, databases, user experience, testing and glue code. Brian is a co-organizer of the Seattle Python Interest Group (www.seapig.org) and has given several talks and tutorials at conferences and user groups.

Technology Requirements: Students must have access to a computer for their assignments and projects, where they can install software (the course does not provide a computer laboratory). It is recommended that students have a portable computer to bring to class. Internet access will be provided at classes.

Assessment criteria: The course is graded Pass/Fail, based on satisfactory completion of required programming assignments and classroom presentations. Attendance is required; more than two unexcused absences will result in a Fail.

Disability accommodation: The University of Washington is committed to providing access and reasonable accommodation in its services, programs, activities, education and employment for individuals with disabilities. For information or to request disability accommodation contact: Disability Services Office: 206.543.6450/V, 206.543.6452/TTY, 206.685.7264 (FAX), or e-mail at dso@u.washington.edu.

Assignments may be completed in groups of 1-3 people. Students are encouraged to actively consult each other and share any relevant reference and support material. However, all code submitted for assignments must be completely written by each group.

Readings

See the Schedule below for specific readings each week.

Reference

Python 2.6 Quick Reference (awesomely dense and complete)
Python 2.6.5 documentation (official documentation from python.org)
Python Standard Library - Internet Protocols and Support
Python Module of the Week - examples of using the modules in the standard library.

Schedule

Tuesdays 6 - 9 pm, January 11 through March 15 (10 sessions).

Topics and readings will be revised before the course begins, consult frequently.

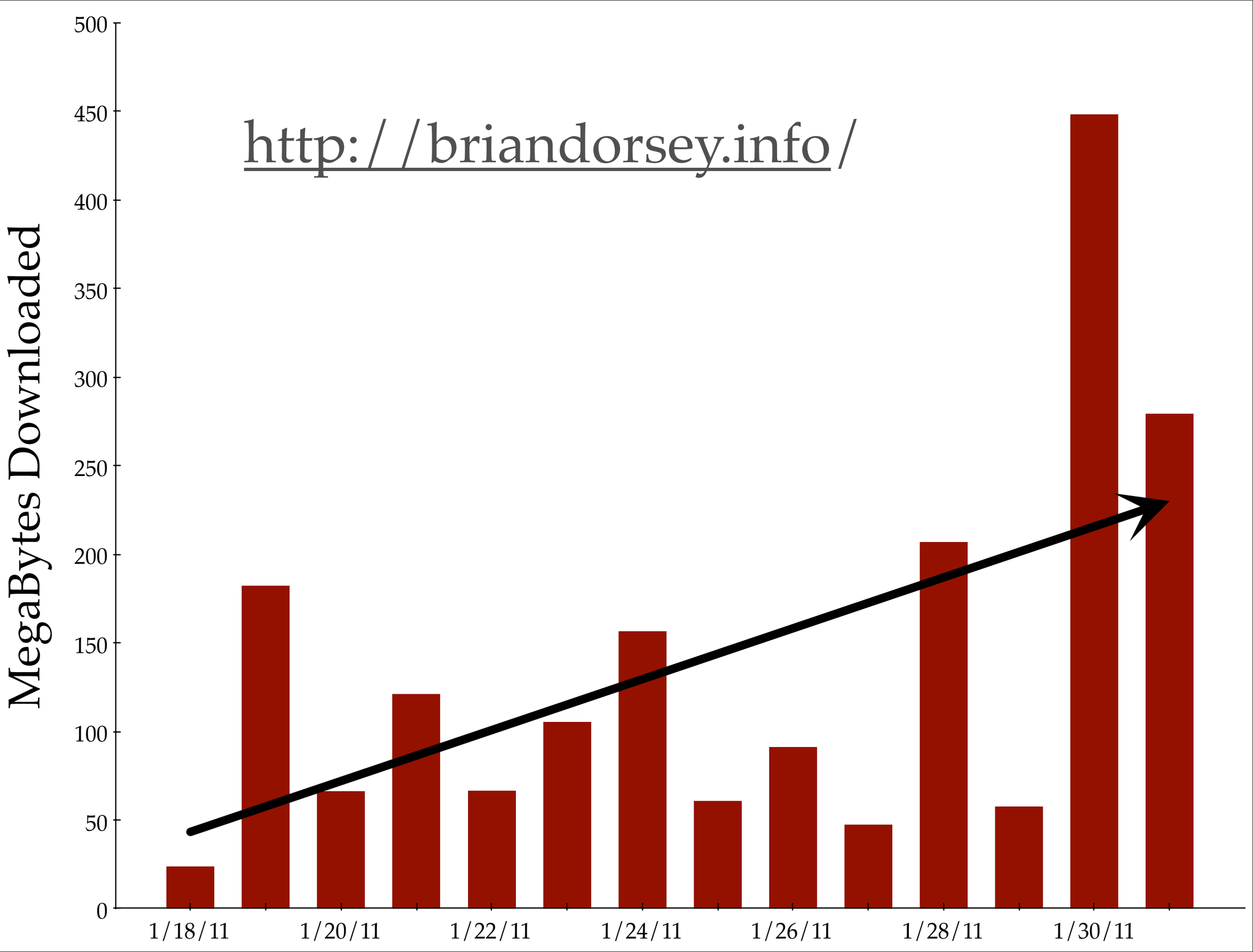
In addition to the topics listed below, each week will include:

- A review of the previous assignment and common problems
- Web API lightning talks: short presentations about useful web services
- Labs: experiment directly with code

Week	Date	Topics	Readings
1	Jan 11	Introductions Networking basics - How computers talk to each other (on the internet). (tcp/ip, sockets, telnet, SMTP) **files:** lecture slides Twilio lightning talk example scripts from the labs	**readings:** WP - Internet Protocol Suite Kessler - TCP/IP (sections 1, 2) WP - Domain Name System WP - Internet Socket RFC 5321 - SMTP - (Appendix D ONLY) (note: S is a server message, C is a client message.) **reference:** (skim before class, needed for labs & assignments) python lib - socket Socket Programming HOWTO python lib - smtplib **extra:** WP - Berkeley socket interface RFC 821 - SMTP (initial) RFC 5321 - SMTP (latest) **bonus:** ZeroMQ Guide - Chapter 1
2	Jan 18	More protocols - Languages of the internet. (POP3/IMAP, FTP, HTTP, others) Guest Speaker: Brian, wearing a different hat. Topic: 30 minute web server	**readings:** Read through this list of modules. If you don't know what a protocol is for, look it up online. Think about their relationship to each other, which are clients? Which are servers? Which clients talk to which servers? Python Standard Library Internet Protocols An introduction to the HTTP protocol: HTTP Made Really Easy **reference:** (skim before class, needed for labs & assignments) ftplib httplib urllib **extra:**

B

http://briandorsey.info/



(example)

reliance

Lecture A (20)

BLUE

www.blueboxgrp.com

rackspace
MANAGED HOSTING

linode.com

slicehost

amazon.com

WebFaction

DreamHost

- Shared: DreamHost, etc, etc
- WebFaction - shared, but you get your own Apache processes
- VM only: Amazon EC2, Slicehost, Linode, etc, etc
- Hybrid: Bluebox, Rackspace
- Colocation: rent space, own servers

recommendation:

not sure what you need?
start with WebFaction

virtual machines

local vs hosted

- VMware (Windows, Linux, Mac)
- Parallels (Mac)
- VirtualBox (Windows, Linux, Mac)
- KVM & many Linux specific options
- thin OSes: Xen, VMware ESX, etc

local dev images are very useful

Windows or Linux

http://blog.restbackup.com/how-to-use-amazon-ec2-as-your-desktop

$26.26 / month

~$315 / year

if you're feeling adventurous, and you want to play with desktop Linux, those instructions should work on our VMs as well

recommendation:

not sure what you need?

try VirtualBox

or

buy VMware

shells

sh, bash, ksh, zsh

command.com, cmd.exe

etc, etc

today: bash

at the level we're going to talk about, most of it works on Windows, too (cmd.exe)

all of it works on Mac

bash is default on OS/X

you can run bash on Windows

www.cygwin.com

how to run shells?

native console on Linux

usually, using a GUI terminal emulator application

recommendation:
(Windows)

PuTTY

recommendation:
(Mac)

iTerm2

recommendation:
(Linux)

use the default terminal app

demo

(login to VM with SSH, mention keys)

ls
ls -l
ls -a
ls -al
tab key!!!!
man
working directory - very significant for python apps
pushd/popd instead of cd
cd by itself goes to home dir
'hidden' file names .something

demo

```
touch
cp
mv
mkdir
rmdir
(names with spaces/funny characters?)
(doublequotes)
```

demo

```
(pipes)
echo
cat
|
>
<
>>
```

demo

```
/home
/home/username
/etc
/var
/tmp
```

more details: http://linuxcommand.org/lts0040.php

demo

```
(files)
touch
file
symbolic links
hard links
```

demo

```
sudo make me a sandwhich
sudo restart
sudo vim /etc/passwd
```

demo

```
ls -al
files have an owner and a group
whoami
rwx - read, write, execute
chmod
    0777 style
    ugoa - user, group, other, all
    +-= - add, remove, set
/etc/passwd
/etc/group
chown
    sudo chown owner:group
chgrp?
symbolic links - permissions are checked and set on
the real file
```

questions?

environment variables

demo

```
set
export VARIABLE=SOMETHING
set VARIABLE=SOMETHING  (on Windows)
echo $VARIABLE  (use in any further commands)
(uppercase by convention)

.bashrc, .bash_profile, .profile
```

Lab A (20)

`git clone git://github.com/briandorsey/uwpython_web.git`
`(or, if you already have a copy: git pull)`

Lab A

- configure your **`EDITOR`** environment variable to point to an editor you like
- create a **`~/bin`** dir, log out and in, verify it is in **`PATH`**, create python script in **`bin`**.
- get comfortable with screen, start it, open multiple windows, switch between them, start editing some text files, close your terminal, reconnect and resume session

Break

(10)

Guest Speaker

(20 +10)

job control

demo

```
xeyes
xeyes &
^z
ps
ps -ejH
ps -ejH
bg
fg
jobs
kill
kill -9
```

some applications

demo

```
sudo apt-get install links
links briandorsey.info/uwpython

grep

nano

vim
```

screen

window manager for shells

hosts shells between logins

demo

```
screen


(play with multiple shells)
^a   (all commands start with control-a)
^ac  (create new shell)
^a^a (switch to last active shell)
^a?  (help - you will need this)
^a"  (list shells)
```

demo

```
(screen - detached)
(disconnect demo)

^ad  (detach on purpose)

screen -list  (show detached screens)
screen -d -r [...] (reattach)
```

moving files around

demo

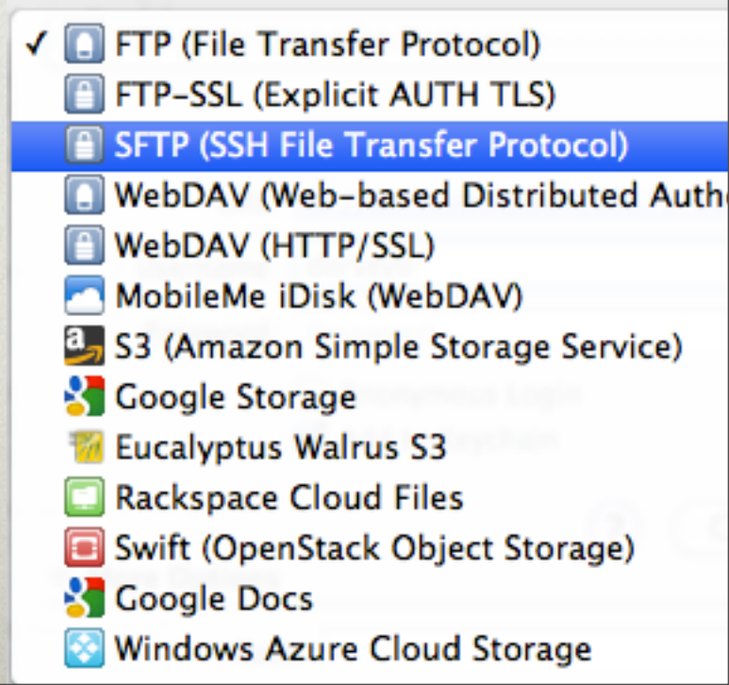
```
scp a_file dorseyb@block115379-pwc.blueboxgrid.com:
(install cygwin or command line ssh tools on Win)
```

recommendation:
(Windows)

WinSCP

recommendation:
(Mac)

CyberDuck



recommendation:
(python)

paramiko

best?

source control

exit

API Talks (20)

talks

Break

(10)

RANDOM

Lecture B (20)

source control

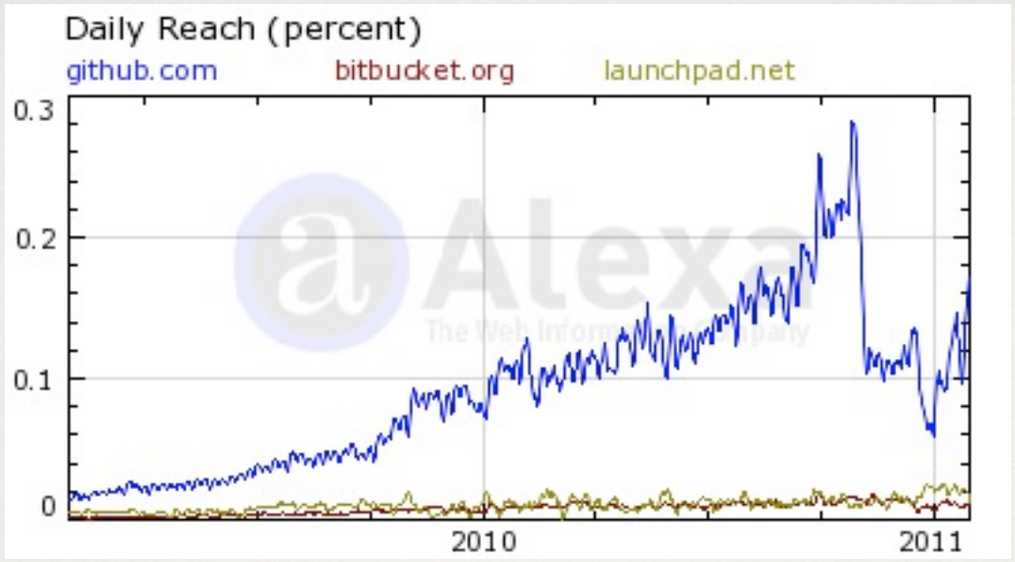
- backup
- backup in logical increments
- log
- deployment / multiple machines
- free your mind
- also...

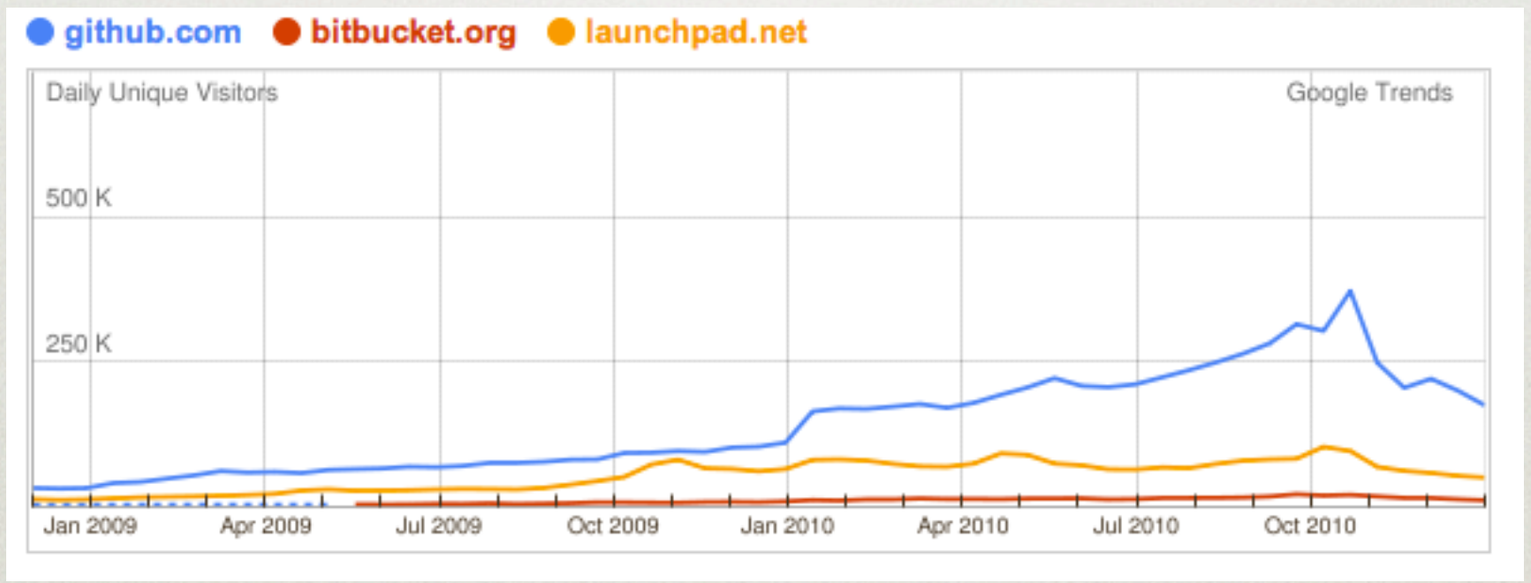
I am a moron.

I'm ignoring lots of really good stuff... but it's there when you're ready to mess with it.
(branches in particular)

why Git?

github.com





http://www.alexa.com/siteinfo/github.com#

http://trends.google.com/websites?q=github.com,bitbucket.org,launchpad.net&geo=all&date=all&sort=0

- mercurial (hg)
  http://mercurial.selenic.com/
- bazaar (bzr)
  http://bazaar.canonical.com/

- Bitbucket - fewer users, free private hosting
  http://bitbucket.org/
- Launchpad - run by Ubuntu, better fit for large projects?
  http://launchpad.net/

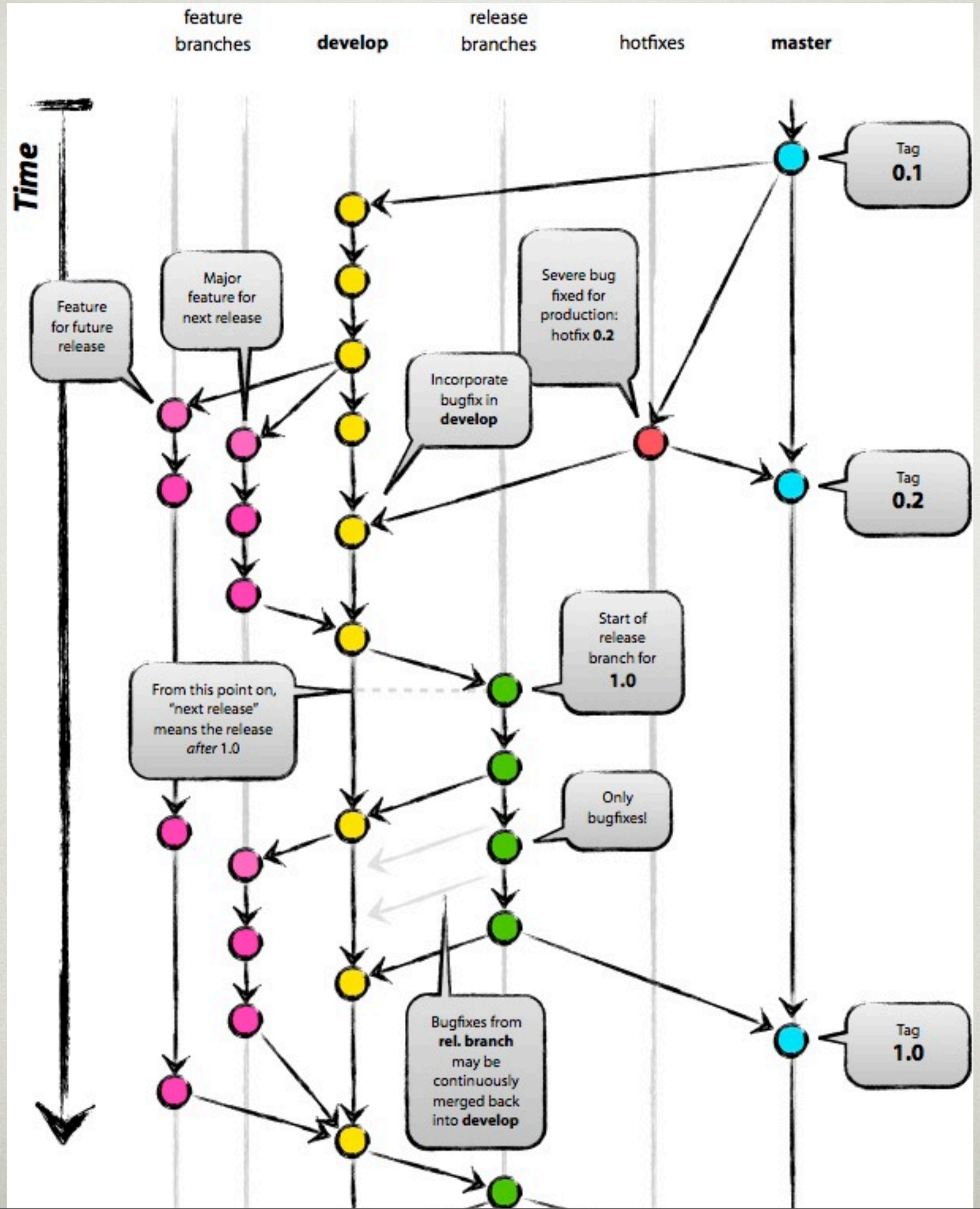
workflow (solo project)

```
(on dev machine)
$ nano some_file
$ git add some_file
$ git commit -m "description of change"
$ git push

(on production)
$ git pull
```

A successful Git branching model (for larger collaboration)

http://nvie.com/posts/a-successful-git-branching-model/


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Ubuntu administration

Ubuntu Server Guide

Web:
https://help.ubuntu.com/10.04/serverguide/C/index.html

PDF:
https://help.ubuntu.com/10.04/serverguide/C/serverguide.pdf

sections:

Package Management

Web Servers

Databases

demo

```
apt-get install
```

daemons (services)

demo

```
sudo /etc/init.d/xyz start
sudo /etc/init.d/xyz stop
sudo /etc/init.d/xyz restart
sudo /etc/init.d/xyz help
```

YOUR OWN PROCESSES

- screen
- cron
- supervisord

Apache
(the web server)

I love this stuff

```
# Do NOT simply read the instructions
# in here without understanding
# what they do.  They're here only as
# hints or reminders.  If you are unsure
# consult the online docs. You have been warned.
```

mess with it anyway

:)

```
/etc/apache2
/etc/apache2/apache2.conf  (ubuntu settings)
/etc/apache2/httpd.conf   (your settings)

a2enmod
a2dismod

htpasswd
(and .htpasswd files)
```

CGI / WSGI

CGI

what is it?

this class will probably be the last time you use it

but, you're going to see echos of it in every web framework in every language

can be hosted by almost any web server

we're going to use two:

CGIHTTPServer
Apache

CGIs output
their own headers,
then a blank line,
then content

Apache CGI config

cgi_test.py

WSGI

what is it?

wsgi_test.py

questions?

Lab B

(20)

- for CGI
- apt-get install libapache2-mod-wsgi
- http://code.google.com/p/modwsgi/wiki/QuickConfigurationGuide
-

`git clone git://github.com/briandorsey/uwpython_web.git`
`(or, if you already have a copy: git pull)`

Lab B

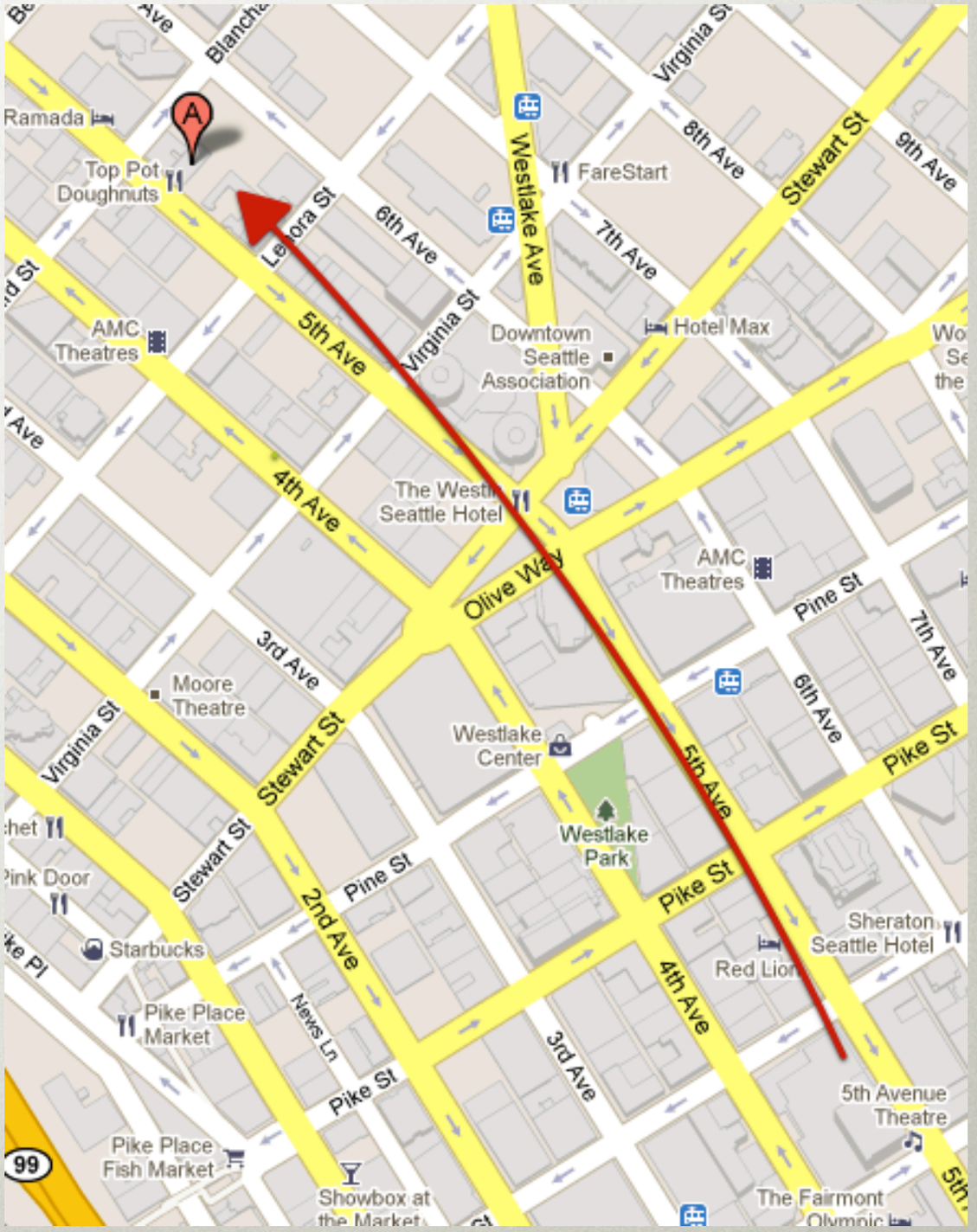
- make sure you can run CGIHTTPServer and **`cgi_test.py`** port your week02 subprocess command line script to a CGI.
- (optional) configure Apache for CGI and run **`cgi_test.py`**
- configure Apache for WSGI and run **`wsgi_test.py`**

WRAPUP

Info

Office hours:
Sunday 2-5pm

Top Pot Donuts
2124 5th Ave
Seattle, WA 98121
206-728-1966



Assignment

```
request:
http://URL?a=135&b=155


your response:
{
    "result": 444,
    "uwnetid": "dorseyb",
    "time": 1296594367.0974219
}

# must work. Verify with: check_assignment.py
# must stay up for 24 hours
```